W05 Sep 16 (D1) Matching sequences detector

Biplav Karna

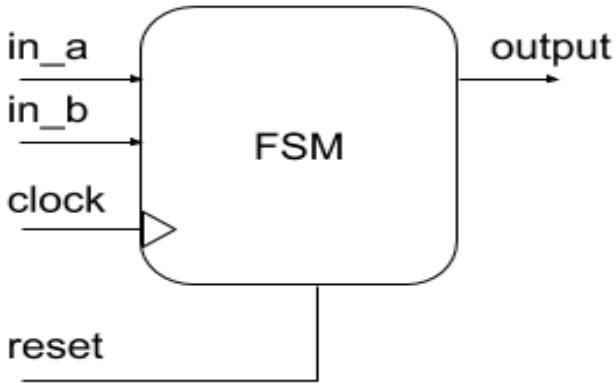
USN Kongsberg

2019

Overview

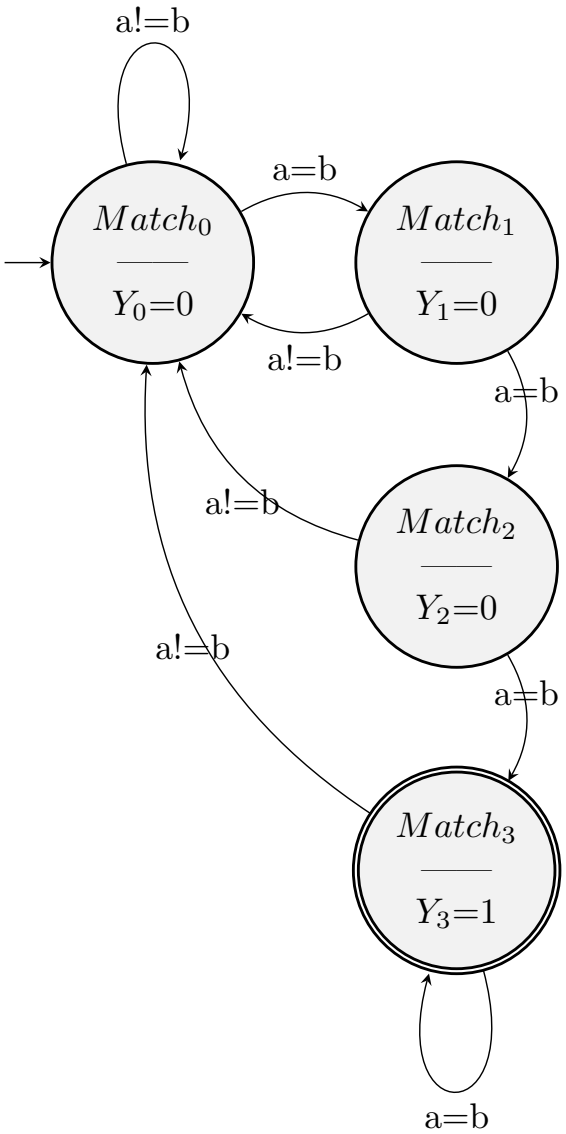
Matching Sequence Detector

Consider a two-inputs, one-output sequence detector that asserts its output whenever the two inputs have the same logic value during the last three consecutive clock cycles (see the example below).

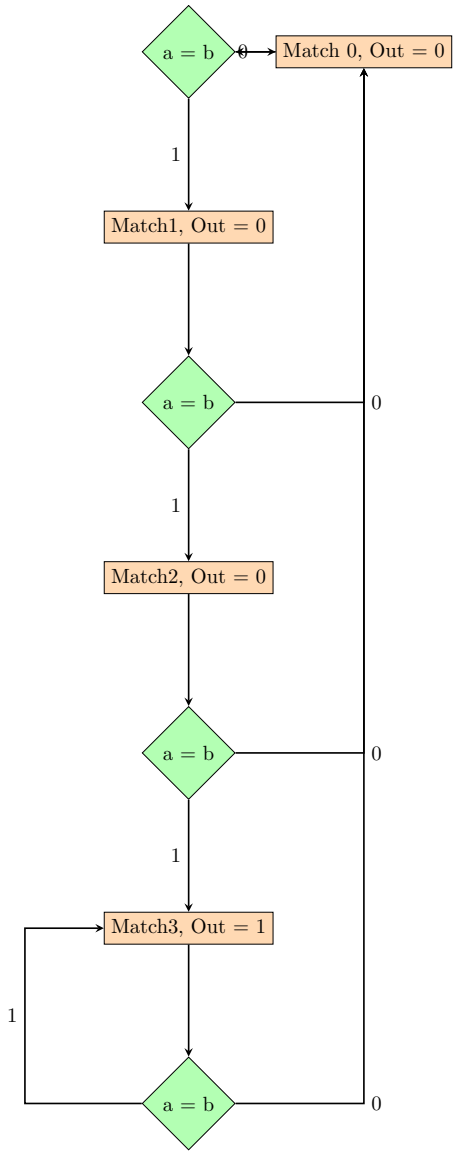


1. Present a state diagram and ASM chart that represent the behaviour of this sequence detector.
2. Build a corresponding VHDL description and demonstrate the correct operation of your solution by simulation.

State Diagram



Asm Diagram


a = b
0
Match 0, Out = 0
1
Match1, Out = 0
a = b
0
1
Match2, Out = 0
a = b
0
1
Match3, Out = 1
1
a = b
0

dff.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dff is
    port (
            rst: in std_logic;
            clk: in std_logic;
            d: in std_logic;
            q: out std_logic
        );

end dff;

architecture dff_arch of dff is


begin

process (rst,clk)
begin
    if (rst = '1') then
        q <= '0';
    elsif ( rising_edge(clk) ) then
        q <= d;
    end if;

end process;

end dff_arch;
```

one-bit-comparator.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity one_bit_comp is
    port (
            i0: in std_logic;
            i1: in std_logic;
            comp_out: out std_logic
        );

end one_bit_comp;

architecture one_bit_comp_arch of one_bit_comp is

begin

comp_out <= (i0 and i1) or ( (not i0) and (not i1)) ;

end one_bit_comp_arch;
```

seq_detect_fsm.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity seq_det_fsm is
    port (
            clk: in std_logic;
            a,b: in std_logic;
            rst: in std_logic;
            result_out: out std_logic
        );

end seq_det_fsm;


architecture seq_det_fsm_arch of seq_det_fsm is

signal fout_a,fout_b: std_logic;
signal comp_result: std_logic;

type seq_state_type is ( match0, match1, match2, match3);
signal state_reg, state_next: seq_state_type;

begin

a_register: entity work.dff(dff_arch) port map( rst =>rst, clk=>clk, d=>a, q=>fout_a);
b_register: entity work.dff(dff_arch) port map( rst =>rst, clk=>clk, d=>b, q=>fout_b);
comparator: entity work.one_bit_comp(one_bit_comp_arch) port map(i0 =>fout_a , i1=>
     fout_b , comp_out =>comp_result);

-- state register section


process (rst,clk)
begin
```

```
    if (rst = '1') then
        state_reg <= match0;
    elsif ( rising_edge(clk) ) then
        state_reg <= state_next;
    end if;

end process;

-- end state register section

-- fsm next state section

process (state_reg,comp_result)
begin
    case state_reg is
        when match0 =>

            if( comp_result = '1' ) then
                state_next <= match1;
            else
                state_next <= match0;
            end if;

        when match1 =>

            if( comp_result = '1' ) then
                state_next <= match2;
            else
                state_next <=match0;
            end if;

        when match2 =>

            if( comp_result = '1' ) then
                state_next <=match3;
            else
```

```
                state_next <=match0;
            end if;

        when match3 =>

            if( comp_result = '1' ) then
                state_next <=match3;
            else
                state_next <=match0;
            end if;

    end case;



end process;


--- end fsm next state section



--- output section

result_out <= '1' when (state_reg = match3) else
              '0';

--- output section
end seq_det_fsm_arch;
```

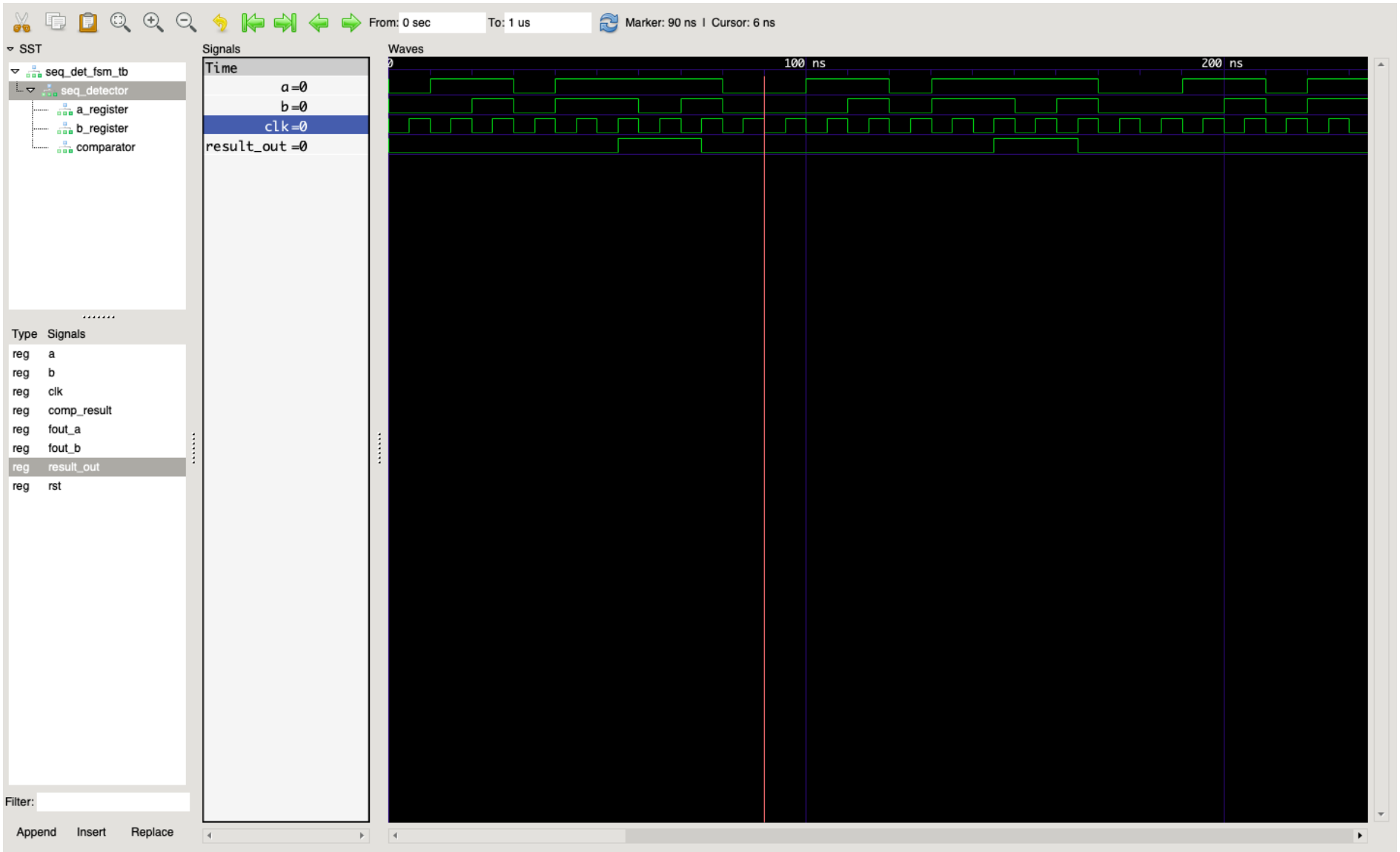
Source (seq_dectect_fsm_tb.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity seq_det_fsm_tb is
end seq_det_fsm_tb;


architecture seq_det_fsm_tb_arch of seq_det_fsm_tb is

constant CLK_PERIOD: time :=10 ns;

signal clk,rst,a,b,result_out: std_logic;
begin

seq_detector: entity work.seq_det_fsm(seq_det_fsm_arch) port map ( clk=>clk, rst=>rst,a=>
    a,b=>b, result_out=>result_out);

clock: process
begin
    clk <= '0';
    wait for CLK_PERIOD/2;
    clk <= '1';
    wait for CLK_PERIOD/2;
end process;


io_test: process
begin
    rst <= '0';
    a <= '0';
    b <= '0';
    rst <= '1';
```

```
wait until falling_edge(clk);
rst <= '0';
a <= '1';
b <= '0';

wait until falling_edge(clk);

a <= '1';
b <= '1';

wait until falling_edge(clk);
--wait for CLK_PERIOD * 4;

a <= '0';
b <= '0';

wait until falling_edge(clk);
a <= '1';
b <= '1';

wait until falling_edge(clk);
a <= '1';
b <= '1';

wait until falling_edge(clk);
a <= '1';
b <= '0';

wait until falling_edge(clk);
a <= '1';
b <= '1';

wait until falling_edge(clk);
a <= '0';
b <= '0';
```

```
    wait until falling_edge(clk);
end process;

end;
```

Output


From: 0 sec
To: 1 us
Marker: 90 ns | Cursor: 6 ns
SST
seq_det_fsm_tb
seq_detector
a_register
b_register
comparator
Type Signals
reg a
reg b
reg clk
reg comp_result
reg fout_a
reg fout_b
reg result_out
reg rst
Filter:
Append Insert Replace
Signals
Time
a =0
b =0
clk =0
result_out =0
Waves
0
100 ns
200 ns

Thanks